Real-time path planning using a modified A* algorithm

Pedro Costa, A. Paulo Moreira, Paulo Costa

***Abstract*— Real-Time path planning is a key issue for the performance of a mobile robot. In this paper, a modified A* algorithm that can plan in real-time the best path is presented. The suggested modifications to the A* algorithm enable it to deal with non static obstacles in an efficient way. It is shown that the proposed algorithm produces better results when used with moving obstacles.**

I. Introduction

Trajectory planning [1] has always been a problem through the times in mobile robotics. This problem can be classified in two cases: when we have a static environment or a dynamic one. The case where the environment is dynamic has an additional difficulty if we can't know the future position of the moving objects.

For a static environment the full trajectory can be planed in advance. For the case where the environment is dynamic and there is some uncertainty on the future position and velocity of some of the obstacles, the trajectory must be re-planed as new information is gained.

The Small Size League of the Robocup Federation (SSL) is an excellent tested for this problem. When two teams of five robots each compete in a robotic soccer game there is a very dynamic environment. While the position, at each instant, for all the robots can be known, the future position for the robots from the opposing team is uncertain. This means that a lot of the obstacles are moving and their future position can not be pre-computed.

In this case we have robots that can achieve speeds above 2 m/s. So, the path planning algorithm has also some very hard real-time constraints.

There are many possible approaches to this problem. Amongst the most popular are the potential field methods [2] [3] [4] [5], where the robot behaves like a particle immersed in a potential field. The target point acts as an attractive force while the obstacles act as repulsive forces. The combination of this influence should lead the robot to the target while avoiding the obstacles. The biggest problem with this approach is that in a cluttered environment it can result in a impossible or time consuming solution.

There are also the probabilistic approximations like the Rapidly-exploring random tress (RRT) [6] where the roadmap is randomly explored. An improved version ERRT [7] tries to achieve a better performance.

Grid based methods, can, with the movement restricted to the grid slots, find an optimal solution. A few of those algorithms: Dijkstra, A * [8] [9] [10], Wavefront , can find the solution quite efficiently but can only deal with a static environment. There is a optimized variant of the A*, the D * [11] [12] that allows to recalculate less than the entire path in response to discovery of new information.

Here, a different approach is attempted by incorporating some of the dynamics in the way that the obstacles are represented in the cells. The standard path optimization is done following the A* algorithm but the cell representation is modified to incorporate some knowledge about the dynamics associated with the moving obstacles.

First, the standard implementation of the A* algorithm is presented, and then the improved obstacle cell representation is proposed. Finally, the results that show the improved performance and generated trajectory are shown.

Trajectory planning

A*

The A* algorithm works with a cell based map. (fig 1)

For the SSL Robotic field, if the cell size is set to 4cm, as the field dimensions are 4.9 m by 3.9, the grid will have 123 by 98 cells.

Each cell represents a node. Each node can be connected to other nodes and moving from one node to the other has an associated cost (fig 2). In this case, the cost is the metric distance between the cell centers. The A* can calculate the path that minimizes the cost from moving from the starting cell to the target cell.

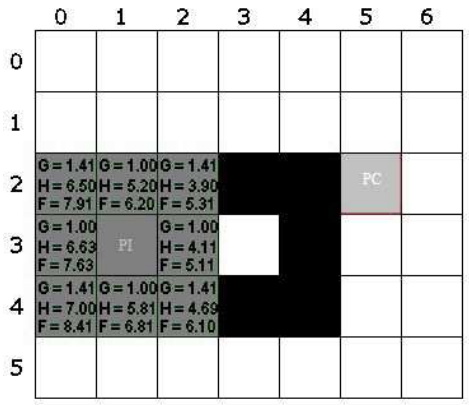


Figure 1 – Cell partitioned environment

Pedro Costa is a PhD Candidate, Paulo Costa and António Paulo Moreira are Assistant Professors with the Department of Electrical and Computer Engineering, Faculty of Engineering of the University of Porto, Rua Dr. Roberto Frias s/n, 4200 465 Porto, Portugal pedrogc@fe.up.pt

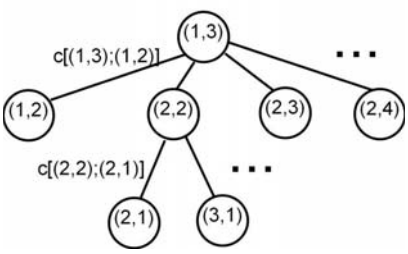


Figure 2 – Associated cost for moving between connected nodes.

The A* Algorithm

There are two lists:

The open list, known as the O-list, that contains the nodes that are candidates for exploration.

The closed list, known as the C-list, that contains the already explored nodes. This nodes where previously in the O-list but as they where explored, they were moved for the C-list.

The nodes in these lists store the "parent" node, which is the node that was used to optimally reach them. This is the node that lies in the shortest path from the origin to current node .

Star(n) – represents the set of neighbors to node n

C(n1,n2) – Cost from going from node n1 to node n2

F(n)=g(n)+h(n) estimate for the lowest cost of going from the origin to the target while passing through node n

g(n) – Cost from the origin to node n

h(n) – An heuristic to estimate the cost of the path from node n to the target node

Algorithm

1. Add origin node to O
2. Repeat
3. Choose $nbest_r$ (best node) from O so that $f(n_{best}) <= f(n)$ $\forall n \in O$
4. Remove nbest from O and add it to C
5. if nbest = taget node then end
6. For all $x \in Q(n_{best})$ which are not in C do:

6.1. if $x \notin O$ then

6.1.1. Adiciona o nó x a O

6.2. else if g(nbest) + c(nbest, x) < g(x) then

6.2.1. Change parent of noce x to n_{best}

7. until O is empty

The basic idea is to choose the best node (lowest cost function) from the O-list. That node is then moved to the C-list and all of its neighboring nodes are processed and inserted in the O-list if they aren't already there. If they are already there, the cost associated with the path from the origin to them is compared with the new one, if the new one is lower, the parent is changed.

This procedure is repeated until the target node is reached or the O-list becomes empty which means that there isn't a feasible solution.

The new cell map construction

The cell map must reflect the possible obstacles affecting the trajectory that the robot must perform. The other robots' velocities can be used to estimate possible collision points.

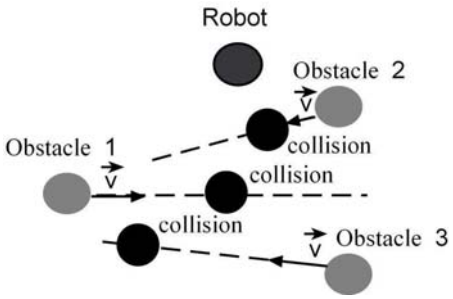


Figure 3 – Collision Points

In this case, while the trajectory must be fully planned, only the first steps are taken before new information arrives and a new calculation is performed. For the SSL team the interval between measures of the robots position is 40 ms. That is also the period of the control loop.
For each calculation the speed of each robot is assumed to be constant. Under that assumption the possible collision point between the robot and an adversary can be estimated. That is where the obstacle will be placed, as it is shown in fig 3

To try to approximate the inherent environment dynamic in a static map there were some techniques that are proposed. They are called:

- Distance
- Slack
- Trail
- Direction

Distance

This change makes the obstacle smaller as the possible collision point is further away from the robot. As the distance increases the relative importance of that obstacle vanishes as is can be seen in figs 4,5.

A distant obstacle mostly does not affect the immediate trajectory points. That can speed up the A* calculation because fewer obstacles will lead to less visited cells and a lower time to find a solution.

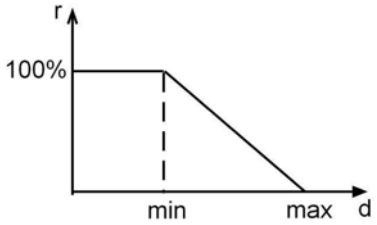


Figure 4: Obstacle size versus distance

In Fig 4 d is the distance between the robot and the estimated collision point, r is the radius of the obstacle, min is the distance above which the obstacle starts losing importance and max is the distance where the obstacle can be discarded.

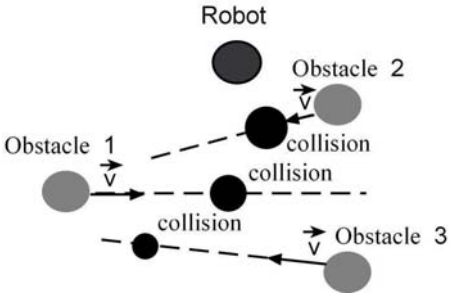


Figure 5: Collision points

Slack

This changes the way an obstacle is represented in the cells. A security area is created around the obstacle. This area is built by setting the cost for those cells above the free ones but still allowing the robot to choose a path through those cells, if the algorithm finds it optimal. This does not make the obstacle bigger but creates a security zone that should be avoided if that does not impact the optimal path. Of course, it can be optimal to use that zone instead of choosing a longer path.

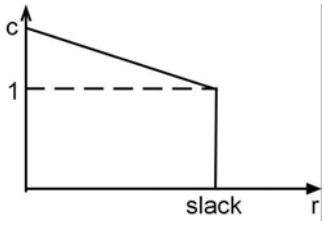


Figure 6 – Slack Zone Cost

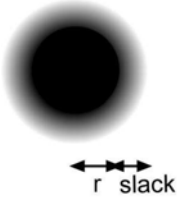


Figure 7 – Obstacle with a slack zone. The black intensity means a higher cost

Trail

A moving obstacle can obstruct the robot for a longer period if the trajectory to avoid the obstacle ends moving the robot parallel to the obstacle movement.

This change creates a certain dynamic awareness to an otherwise static map. It creates an additional zone where the cost to travel there is increased. This zone is created around the projected future positions for the obstacle. The size of this zone depends on the speed of the obstacle. As it is shown in fig 8.

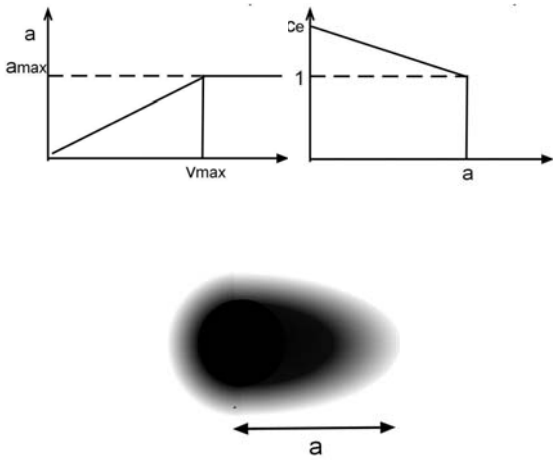


Figure 8 – a) Determination of the size b) Trail Cost
c) Obstacle shape change due to its motion

Direction

This change tries to set the required direction used by the robot as it approaches the target. Without it the robot will hit the target destination from any direction.

There are cases when the approach direction is mandatory. For this case a restriction like in fig 9 is used.

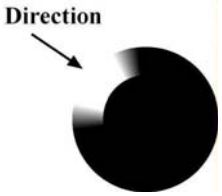
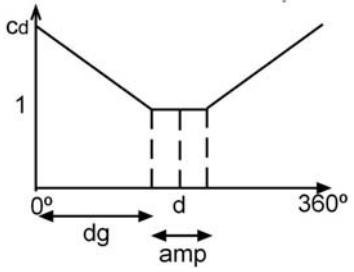


Figure 9 – Target point with mandatory approach direction

Sometimes there is preferred direction but that restriction is not hard. It can violated if the gain in the arrival time is significant. To achieve this, a softer version of the extra obstacle is used, as it can be shown in fig 10



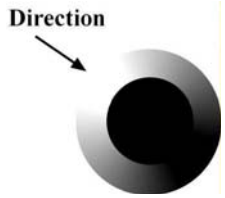


Figure 10 – Target point without hard restriction on the target point

Results

A series of simulations were performed to optimize the parameters for this modification.

A cost function that weights the different performance targets is shown

$$F(x)=0.7*a*T + 0.2*b*P + 0.1*c*C$$

Em que:

T – Time to reach the target.
P - Processing time for the A*.
C – number of colisions
a, b e c – Normalizing factors for different setups

The optimal solution has the robot reaching the target in the shortest time while the algorithm completes in the shortest time also and keeping the collisions as low as possible. The different weighting represents the compromises that are necessary to make. Of course, having the robot reach the target in the shortest time is the most important issue. The processing time must be kept low because that means an extra delay in the control loop and the overall control stability can be compromised. Keeping the collision count low is also desirable.

TABLE I
OPTIMIZED PARAMETERS

Distance	
max	1.25 m
min	0.75 m
Slack	
slack	0.15 m
c	5
Trail	
Ce	5
a	0.65 m
Direction	
amp	60
Cd	5

Two examples were created to test the gains that the new algorithm can yield.

For the first case (fig 11),we have a robot and a obstacle that crosses its path.

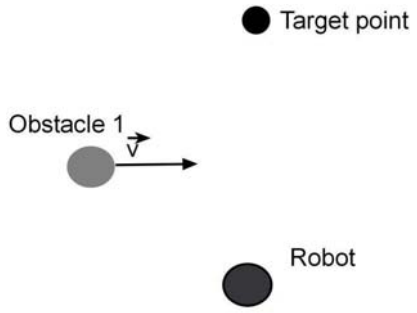
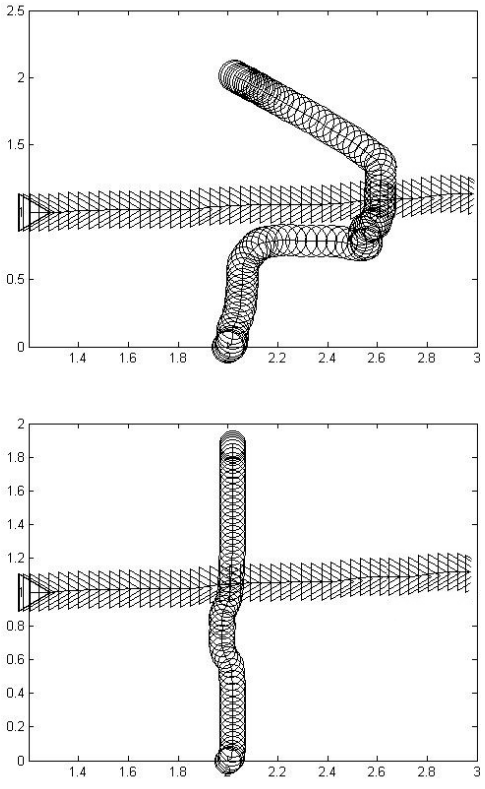


Figure 11 – An obstacle that crosses the robot's path

What happens here is that, without using the obstacle speed, as the robot tries to avoid the obstacle it is dragged in the directions of its movement this happens because the solution where the robot goes around the obstacle choosing to pass in front of it, is the one that seems optimal.



Δ Obstacle
○ Our Robot

Figure 12 – a) Standard b) modified A* results for the case presented in fig 11

TABLE II
RESULTS FOR SITUATION PRESENTED IN FIG. 11

	Standard A*	Modified A*
Time to reach target	2.53s	1.92s
Processing time	0.46ms	0.55ms
Collisions	0	0

In another case that could be taken from a robotic soccer game there are several robots traveling in the field.

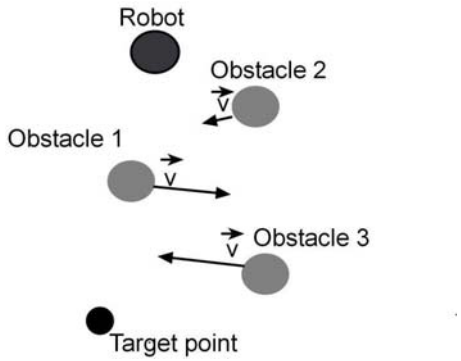
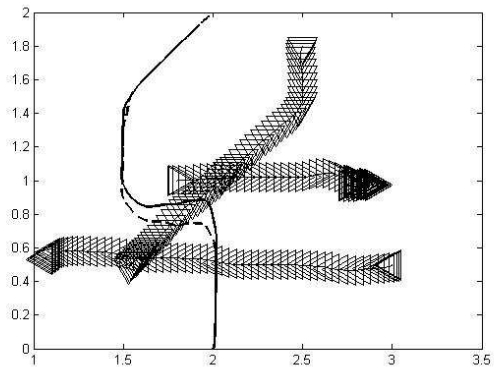


Figure 13 – Several obstacles crossing the robot's path.

For this case the optimal path isn't affected by obstacles 1 and 3, only obstacle 2 will matter.



Δ Obstacles
—— Standard A*
----- Modified A*

Figure 14– Standard and modified A* results for the case presented in fig 13

TABLE III
RESULTS FOR THE CASE PRESENTED IN FIG. 13

	Standard A*	Modified A*
Time to reach target	2.88s	2.76s
Processing time	0.72ms	0.71ms
Collisions	0	0

Comments

In both cases the modified algorithm found a solution where the robot reached the target in less time. The main reason was the drag effect was avoided. In the first case, we were able to avoid the drag of the robot, with the introduction of the trail effect the path chosen no longer to try to go ahead but to pass behind. In the second case, as the obstacle comes into the robot, the path chosen is the one that makes the robot turn up earlier with a smoother path. This is the main gain from using the modified version. Naturally, these solutions are more efficient

The processing time increased in one case and decrease in the other. Three factor work here: the first is to create the modified map increases processing time; second is that the trail effect creates larger obstacles and could increase the number of cells to expand and thus increase the total processing time. The other factor is the shrinkage of obstacles considered far away, that will reduce the processing time.

Both algorithms achieved a trajectory without any collisions.

While this improvements where presented in a robotic soccer setting the advantages that the modified algorithm shows can be found in other cases not specific to robotic soccer competitions. There are many situations where the obstacles are known but their future movement can only be predicted.

The direction restriction was inspired in a typical robotic soccer problem and its contribution is not related with the execution time of the algorithm.

A future improvement should be a way to reflect some dynamical restrictions that the robots have in the obstacles shape to achieve trajectories better suited to the high trajectory speeds.

REFERENCES

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. Kandor, W. Burgard, L. Kavraki, and S. Thurn , " Principles of robot motion theory, algorithms, and implementation " 2005.

[2] J. Borenstein and Y. Koren, " Real-time obstacle avoidance for fast mobile robots," IEEE Transactions On Systems, Man, And Cybernetics, Vol. 19, No. 5, September/ October, pp. 1179-1187 1989.

[3] Hwang, Yong K., "A potential field approach to path planning" IEEE Transactions On Robotics And Automation, Vol. 8, No. 1, February, Pp. 23-32, 1992

[4] E. H. Miller, "Exact Robot Navigation Using Artificial Potencial Functions," *IEEE Trans. On Robotics and Automation, Vol 8, No. 5, October, pp. 501-518, 1992*.

[5] S.S. Ge, Y. J Cui, "Dynamic Motion Planning for Mobile Robots Using Potential Field Method" *Autonomous Robots 13, pp. 2007-222, 2002*

[6] S. M . LaValle " Rapidly-exploring randon trees: A new tool for path planning" , In Technical Report Nº 98-11, October 1998.

[7] James Bruce and Manuela Veloso. Real-Time Randomized Path Planning for Robot Navigation. In *Proceedings of IROS-2002*, Switzerland, October 2002.

[8] T. Goto, T. Kosaka, and H. Noborio, "On the Heuristics of *A** or *A* Algorithm in ITS and Robot Path-Planning" *IEEE Trans.* on Intelligent Robots and Systems, October, pp. 1159-1166, 2003

[9] P. Melchior, B. Orsoni, O. Lavialle, A. Poty, A. Oustaloup, "Consideration of obstacle danger level in path planning using A$*$ and Fast-Marching optimisation: comparative study," Signal Processing 83 pp. 2387 – 2396, 2003.

[10] K. Yamaguchi and S. Masuda, "An *A*$*$ Algorithm with a New Heuristic Distance Function for the 2-Terminal Shortest Path Problem," *IEEE Trans.* Fundamentals, Vol.E89–A, No.2 February, pp. 544-550, 2006

[11] Karen I. Trovato and Leo Dorst, "Differential A*," IEEE Transactions On Knowledge And Data Engineering, Vol. 14, No. 6, November/December, pp. 1218-1229, 2002

[12] D. Cagigas, "Hierarchical *D** algorithm with materialization of costs for robot path planning", Robotics and Autonomous Systems 52, pp. 190–208, 2005